# 优先队列

## 概述

## 实现

优先队列的实现方法包括：

1. 无序数组实现
2. 无序链表实现

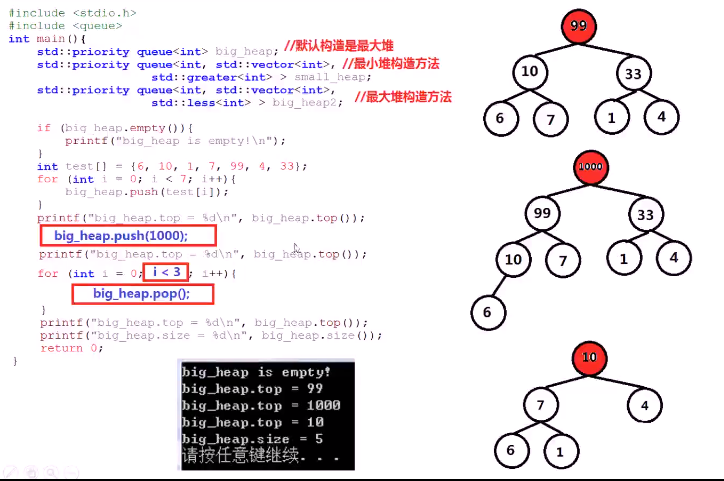
我们可以使用一个简单链表在表头以O(1)执行插入操作，并遍历该链表以删除最小元素，这又需要O(N)时间。另一种方法是，始终让表保持排序状态，这使得插入代价高昂O(N)，而DeleteMin花费低廉O(1)。

1. 有序数组实现
2. 有序链表实现
3. 二叉搜索树实现

二叉查找树对插入和删除操作的平均运行时间都是O(logN)，尽管插入时随机的，而删除则不是。反复除去左子树中的节点似乎损害树的平衡，使得右子树加重。然而，右子树是随机的。在最坏的情形，即DeleteMin将左子树删空的情况下，左子树拥有的元素最多也就是它应具有的两倍。注意，通过使用平衡树，可以把界变成最坏情形的界，这将防止出现坏的插入序列。

1. 平衡二叉树实现
2. 二叉堆实现

二叉堆实现：



## 应用

优先队列应用如下：

压缩数据：霍尔曼编码算法

查找最短路径：Dijkstra算法

计算最小生成树：Prim树

编写选择算法：寻找第k小的元素

# 堆

## 概述

## 分类

### 二叉堆

### d-堆

d-堆是二叉堆的简单推广，它恰像一个二叉堆，只是所有节点都有d个儿子（因此，二叉堆是2-堆）。

### 左式堆

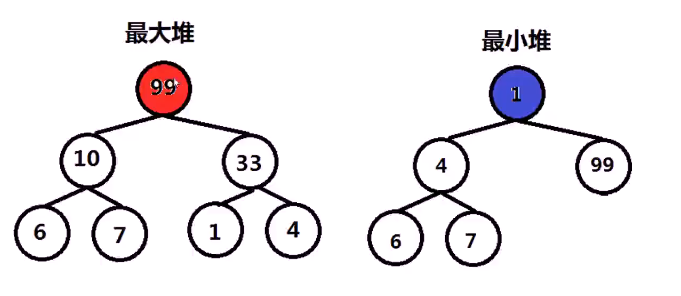
### 斜堆

### 二项队列

## 二叉堆

### 概述

最（大）小二叉堆，最（大）小值先出的完全二叉树。



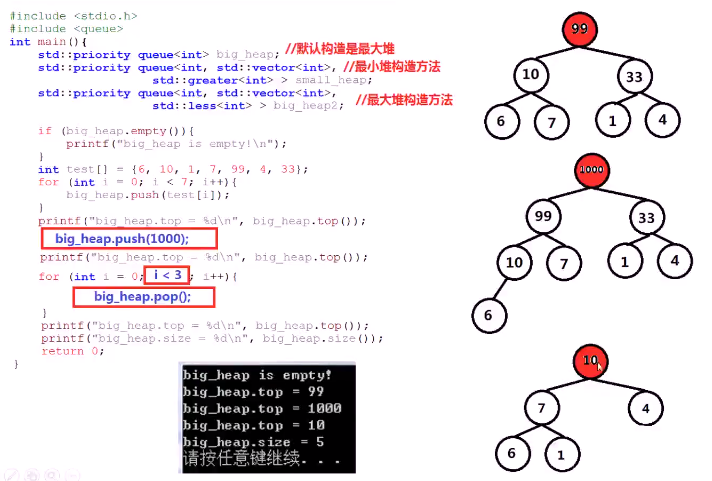
### 表示

### 创建

### 操作

#### 查询

#### 插入



#### 删除

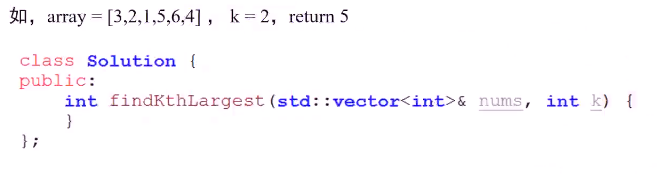
#### 销毁

## 应用

凡是涉及第k个最值问题，通常可以考虑最大/小堆的实现方式。

### 数组中第K大的数

**题目：**已知一个未排序的数组，求这个数组中第K大的数字。



注：Leetcode 215

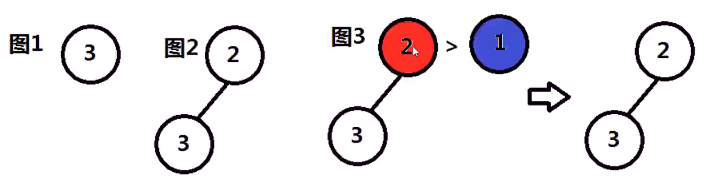
**分析：**

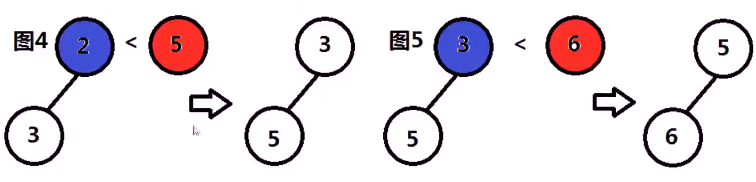
维护一个K大小的最小堆，堆中元素个数小于K时，新元素直接进入堆；否则，当堆顶小于新元素时，弹出堆顶，将新元素加入堆。

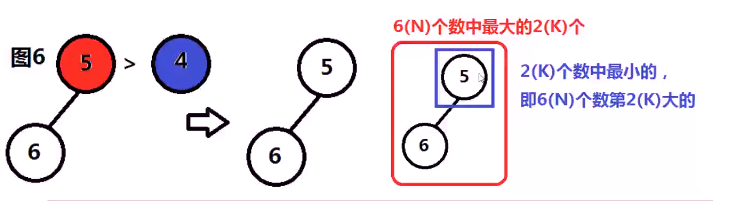
由于堆是最小堆，堆顶是堆中最小元素，新元素都会保证比堆顶小（否则新元素替换堆顶），故堆中K个元素是已扫描的元素里最大的K个；堆顶即为第K大的数。

设数组长度为N，求第K大的数，时间复杂度：N\*logK。

如array=[3,2,1,5,6,4]：





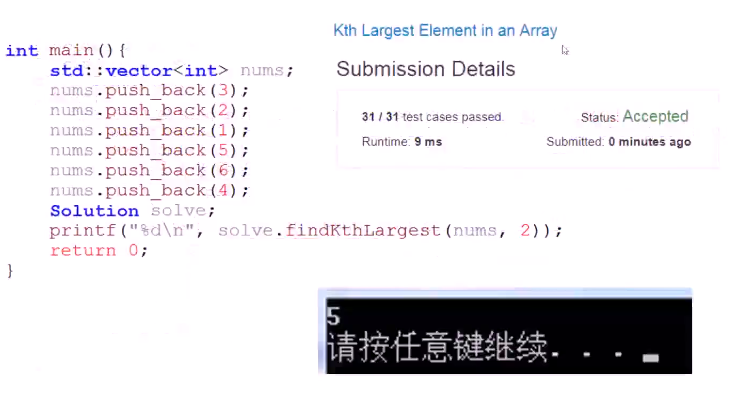


**代码：**



注：掌握最小堆的构建方式，即priority\_queue<>

**测试：**



### 最接近原点的k个点

### 找出第k小的距离对

### 查找和最小的k对数字

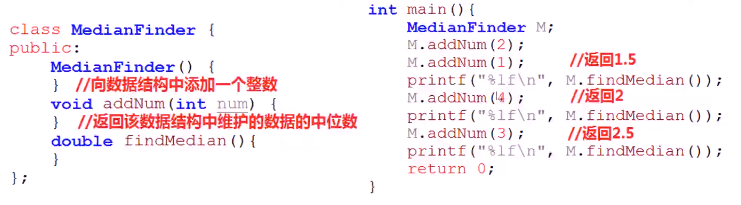
### 寻找中位数

题目：设计一个数据结构，该数据结构动态维护一组数据，且支持以下操作：

1. 添加元素：void addNum(int num)，将整数num添加至数据结构中；
2. 返回数据的中位数：double findMedian()，返回其维护的数据的中位数。

中位数的定义：

1. 若数据个数为奇数，中位数是该组数排序后中间的数。[1,2,3]🡪2
2. 若数据个数为偶数，中位数是该组数排序后中间的两个数字的平均值。[1,2,3,4]🡪2.5



注：Leetcode 295

**分析：**

最直观的方法：

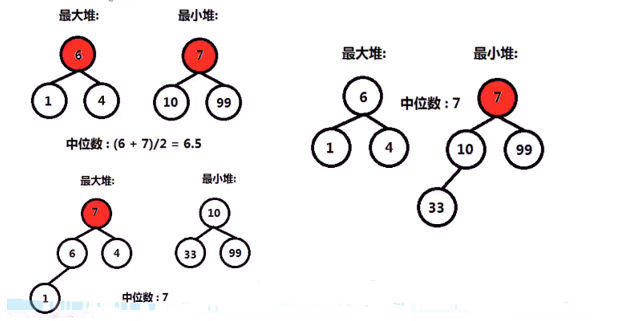
存储结构使用数组，每次添加元素或查找中位数时对数组排序，再计算结果。

时间复杂度：

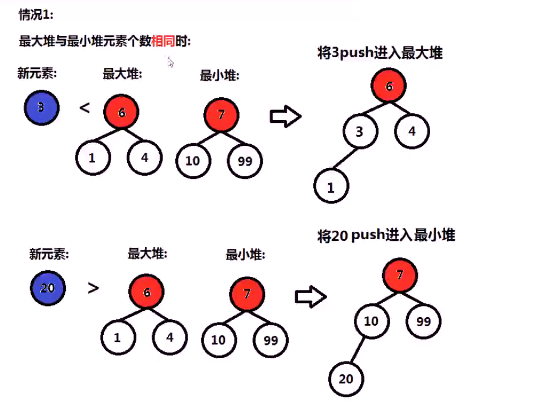
1. 若添加元素时排序，addNum复杂度O(n)，findMedian复杂度O(1)
2. 若查询中位数时排序，addNum复杂度O(1)，findMedian复杂度O(nlogn)

若添加元素或查询中位数是随机的操作，共n次操作，按照上述思想，整体复杂度最佳为O(n^2)。

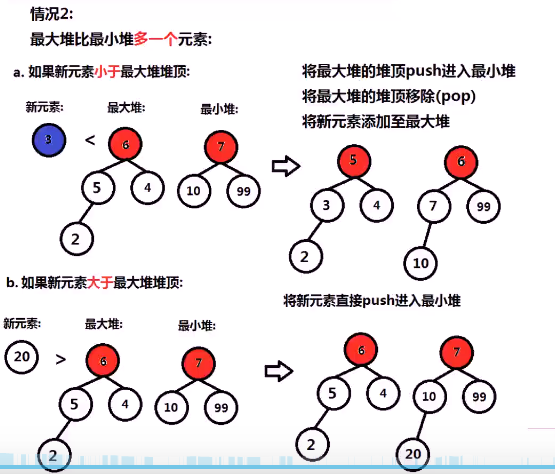
除了上述方法，还可以动态维护一个最大堆和一个最小堆，最大堆存储一半数据，最小堆存储一般数据，维持最大堆的堆顶比最小堆的堆顶小。



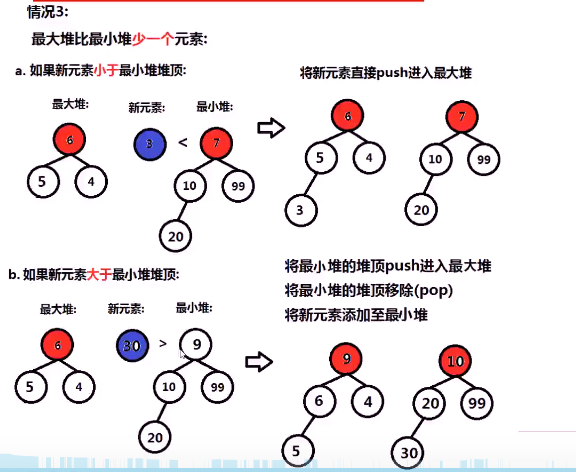
情况1：最大堆与最小堆元素个数相同时：

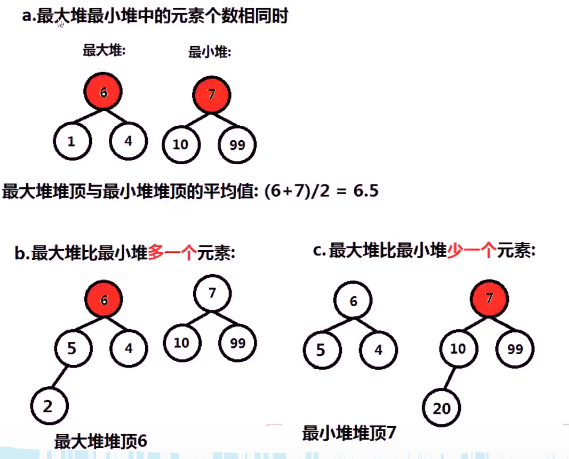


情况2：最大堆比最小堆多一个元素：



情况3：最大堆比最小堆少一个元素





**代码：**

